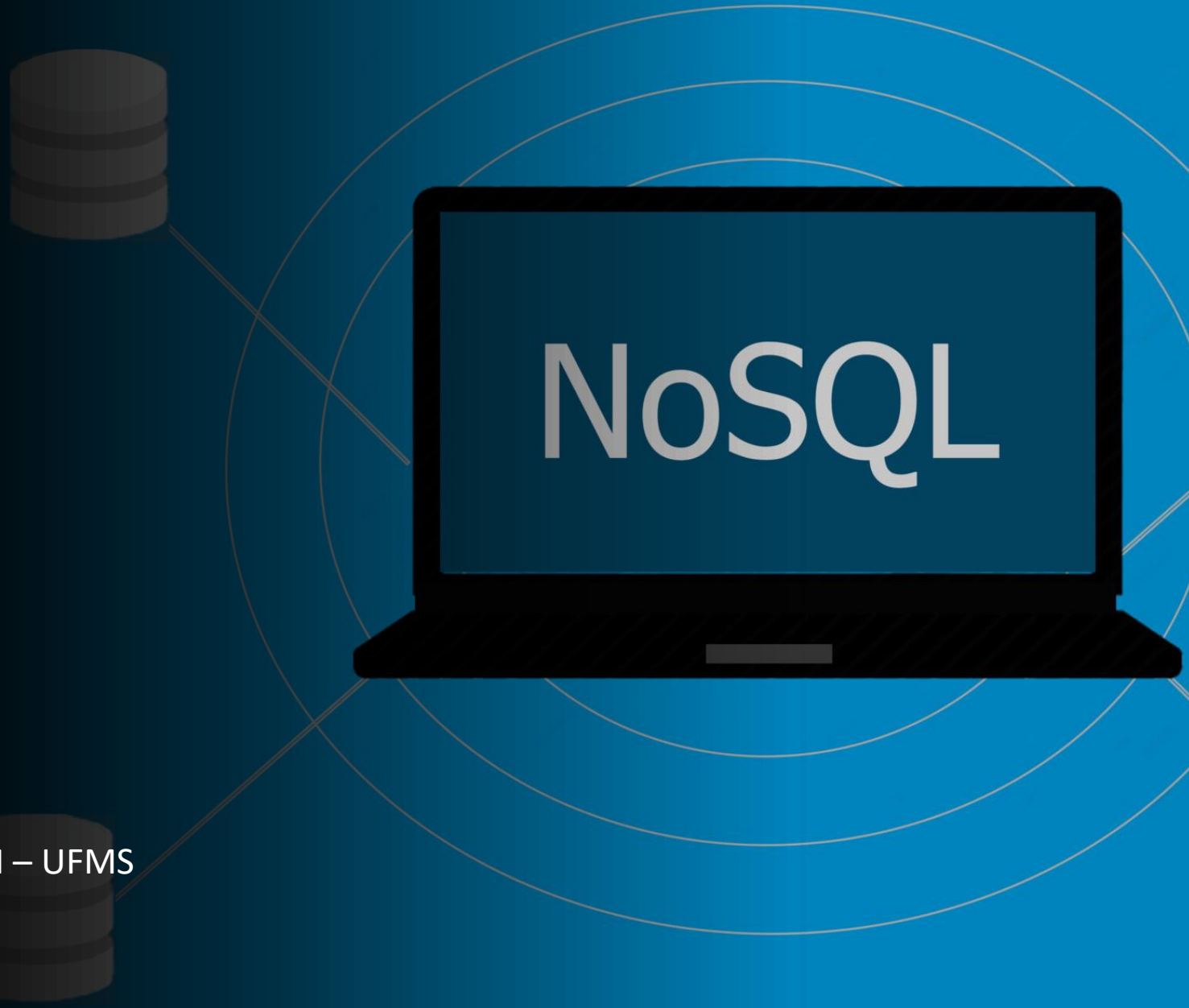
# Introdução ao NoSQL




Vanessa Araujo Borges
Laboratório de Banco de dados – FACOM – UFMS

# Perspectiva histórica

NoSQL
Not Only SQL

- NoSQL (*Not Only SQL* – não somente SQL) é um termo genérico definido para uma classe de banco de dados **não relacionais (ACID)**
  - O termo iniciou em 1998 como o nome de um banco de dados relacional de código aberto que não possuía uma interface SQL
  - Em 2009 o termo foi utilizado por Eric Evans como nome de um evento para discutir **bancos de dados *open source* distribuídos**
    - Essa foi uma tentativa de descrever o surgimento de um número crescente de banco de dados não relacional que não tinham a preocupação de fornecer garantias ACID

# Banco de dados NoSQL

NoSQL
Not Only SQL

- Foco no armazenamento de **grandes volumes de dados** (big data)
- Características
  - Não relacional
  - Altamente escalável
  - Eficientemente distribuído
  - Em geral, de código aberto
  - Replicação facilitada dos dados
  - SQL não é a linguagem padrão
  - Alto desempenho
  - Livre de esquema
  - ...

# Banco de dados NoSQL

- Grandes empresas mundiais utilizam e apoiam o NoSQL
- O Google investe desde 2004 no BigTable, um banco desenvolvido para suprir as necessidades de armazenamento da empresa
- O Facebook projetou o Cassandra, um sistema de alta disponibilidade e escalabilidade, para trabalhar com o seu grande fluxo de informações.
  - Seu código fonte foi disponibilizado para a comunidade e atualmente é mantido pela apache.

- **Bancos de dado nosql não possuem como objetivo subutilizar os Bancos de Dados Relacionais, e sim apresentar uma alternativa**

# Banco de dados NoSQL

- Existem dezenas de bancos NOSQL porque existem dezenas de problemas de persistência de dados que o SQL tradicional não resolve.

- Bancos não-relacionais document-based (que armazenam seus dados em documentos) são os mais comuns e mais proeminentes de todos, sendo o seu maior expoente o banco MongoDB
  - Pesquisa mais recente de bancos de dados utilizados pela audiência do StackOverflow em 2017 mostra.

# Porque NoSQL?

- O crescente uso de **aplicativos online** resultou em um número crescente de operações de banco de dados e uma necessidade de uma maneira mais fácil de escalar bancos de dados para atender a essas demandas
- É necessário uma solução altamente flexível, que acomode facilmente qualquer novo tipo de dado (**não-estruturado e semi-estruturado**) e que não seja corrompida por mudanças na estrutura de conteúdo
- **IoT:** tecnologia NoSQL para dimensionar o acesso simultâneo de dados para empresas inovadoras estão utilizando milhões de dispositivos e sistemas conectados, armazenar bilhões de pontos de dados e atender aos requisitos de infraestrutura e operações de missão crítica de performance
- **Cloud computting:** atualmente a maioria dos novas aplicações são executados em um sistema em nuvem privado, público ou híbrido, suportam um grande número de usuários

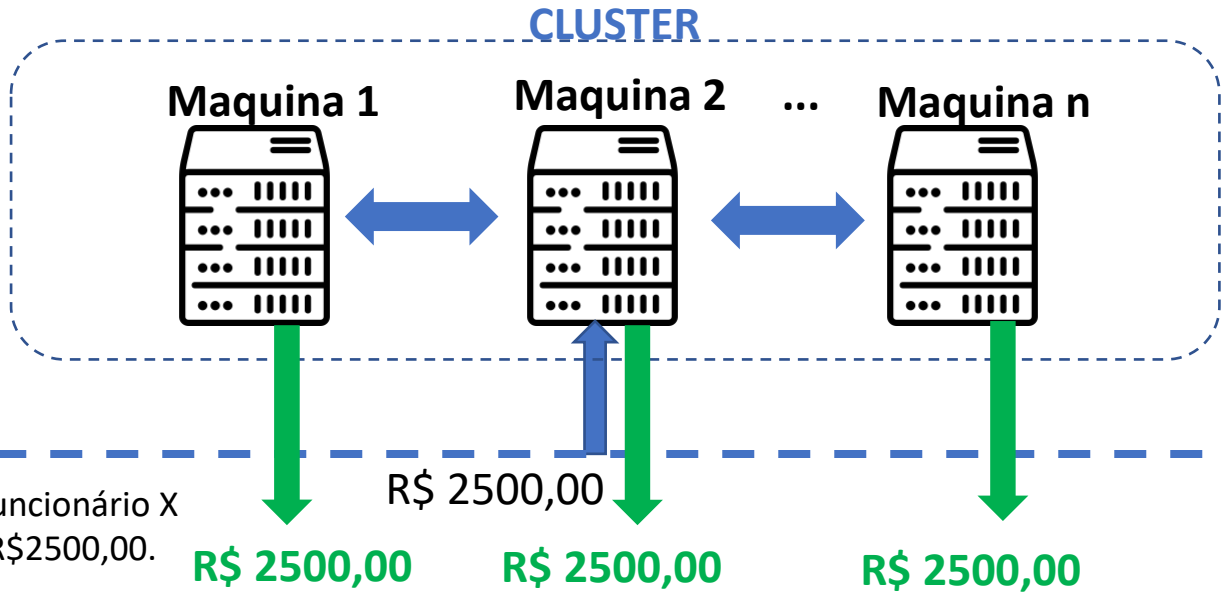
# Relacional *versus* NoSQL

| | Banco de dados relacional | Banco de dados NoSQL |
|---|---|---|
| Tipo de dados | Estruturado | Semiestruturado ou não estruturado |
| Escalabilidade | É possível, porém possui uma certa complexidade, o que dificulta a implementação. Quanto mais dados, mais espaço no servidor e memória são necessários. | Como não possui estrutura fixa, a inclusão de dados é bastante flexível. Quanto mais dados, aumenta-se o número de servidores, que podem ser ou não de alta performance, barateando e otimizando o armazenamento. |
| Disponibilidade | Uma grande demanda pode ser prejudicial(dificuldade com o volume de acesso e distribuição de dados). | Possui um alto grau de distribuição de dados (em vários servidores), e garante um maior número de solicitações. |

# Relacional *versus* NoSQL

| | Banco de dados relacional | Banco de dados NoSQL |
|---|---|---|
| Característica | Tabelas, esquema definido, redundância mínima, entidade e relacionamento, ACID | Registros, não possui estrutura fixa, escalabilidade |
| Necessidades | Sistemas financeiros, sistemas corporativos, consistência dos dados | Sistemas em nuvem, análises sociais, performance na consulta/escrita, replicação |
| SGBD | DB2, Firebird, InterBase, SQL Server, MySQL, Oracle, PostgreSQL | Cassandra, BigTable, MongoDB, CouchDB, Dynamo |
| Empresas que utilizam | SAP, OpenERP, Previdência Social, Caixa, Itaú | Twitter, Facebook, Amazon, LinkedIn, Google, Yahoo |
| Consistência | As regras de integridade garantem a consistência das informações. **Consistência rígida** | Não garante a consistência da informação, caso nenhuma informação seja atualizada, retornará o mesmo valor para todas as solicitações. **Consistência eventual** |

# Consistência rígida




**Tempo 1:**
Altere o salario do funcionário X de R$2000,00 para R$2500,00.

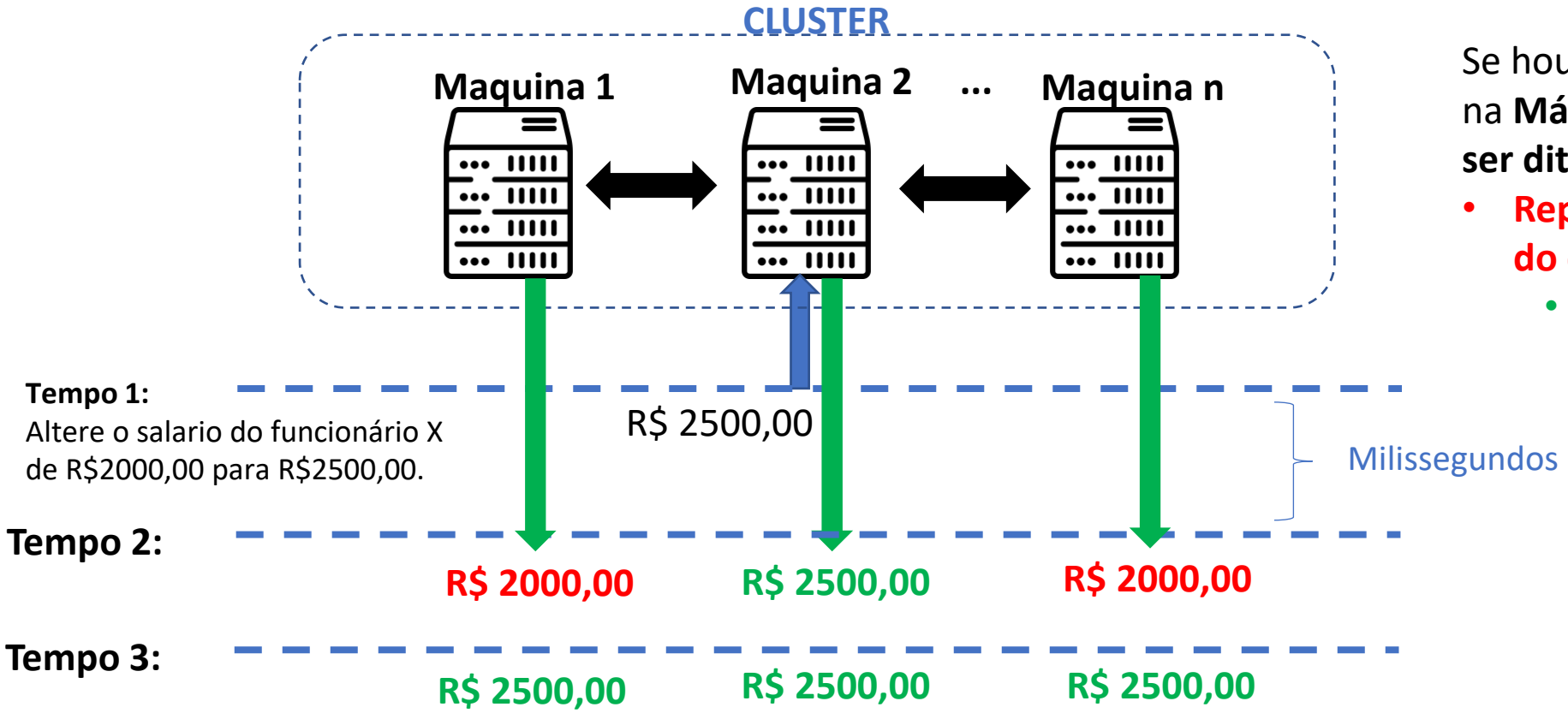
- **Visão única dos dados**
- **Dados CONSISTENTES**

**Banco de dados relacional**

Se houver uma alteração de salário na **Máquina 2**, essa alteração só será persistida **quando todos os nós do cluster forem atualizados**

- **Demanda grande esforço de replicação de dados entre os nós do cluster**
  - **Problemas de escalabilidade**

# Consistência eventual (assíncrona)




Se houver uma alteração de salário na **Máquina 2**, **essa alteração pode ser dita concluída**

- **Replicação dos dados entre nós do cluster de forma assíncrona**
  - **Simplifica escalabilidade**

**Banco de dados nosql**

# Consistência eventual

- Considere uma operação de escrita *op1* no tempo *t1* que ocorreu no nó *n1* e que alterou o dado do valor *x* para *y*, e uma operação de leitura *op2* no tempo *t2* que foi atendida pelo nó *n2*.
- Com base nas propriedades BASE, é possível que a operação *op2* tenha como resposta o valor *x* (valor antigo) ao invés do valor *y* (novo valor), dependendo do tempo decorrido entre *t1* e *t2*.
- Entretanto, é garantido que se o tempo entre *op1* e *op2* for suficiente, *op2* terá como resposta o valor *y*.

# Transações: ACID


- **Banco de dados NoSQL não garante a propriedade ACID**
- Essa é uma propriedade de transações de **SGBDs relacionais**
  - As transações devem possuir propriedades denominadas **ACID**
  - Essas propriedades devem ser impostas pelos métodos de controle de concorrência e recuperação do SGBD.

    - **Atomicidade**
    - **Consistência**
    - **Isolamento**
    - **Durabilidade ou permanência**

# Propriedade das transações - ACID


- **Atomicidade**
  - Uma transação é uma unidade indivisível (atômica)
  - Garante que todos os efeitos de uma transação em um banco de dados ou nenhum deles é aceito
  - Uma falha não pode deixar o banco de dados em um estado em que a transação é parcialmente executada

- Unidade atômica
- Executa em sua totalidade (nunca parcialmente)

```
read_item(A);
A:=A-50
write_item(A);
```

**Falha!**

Assegurar a **ATOMICIDADE** de uma transação é **responsabilidade do SGBD**, mais especificamente dos componentes de **Gerenciamento de Transações** e de **Recuperação de Falhas**.

# Propriedade das transações - ACID


- **Consistência**
  - Garante que se o banco de dados for inicialmente consistente, a execução da transação (por si só) deixa o banco de dados em um estado consistente


Assegurar a **CONSISTÊNCIA** de uma transação é **responsabilidade do programador.**

# Propriedade das transações - ACID


- **Isolamento**
  - Transações são isoladas umas das outras
  - Garante que **transações executadas concorrentemente sejam isoladas umas das outras**, de modo que cada uma tenha a impressão de que nenhum outra transação está sendo executada concorrentemente a ela

T1

```
read_item(A);
A:=A-50
write_item(A);
read_item(B);
B:=B+50
write_item(B);
```

T2

```
read_item(A);
A:=A*1,5;
write_item(A);
```

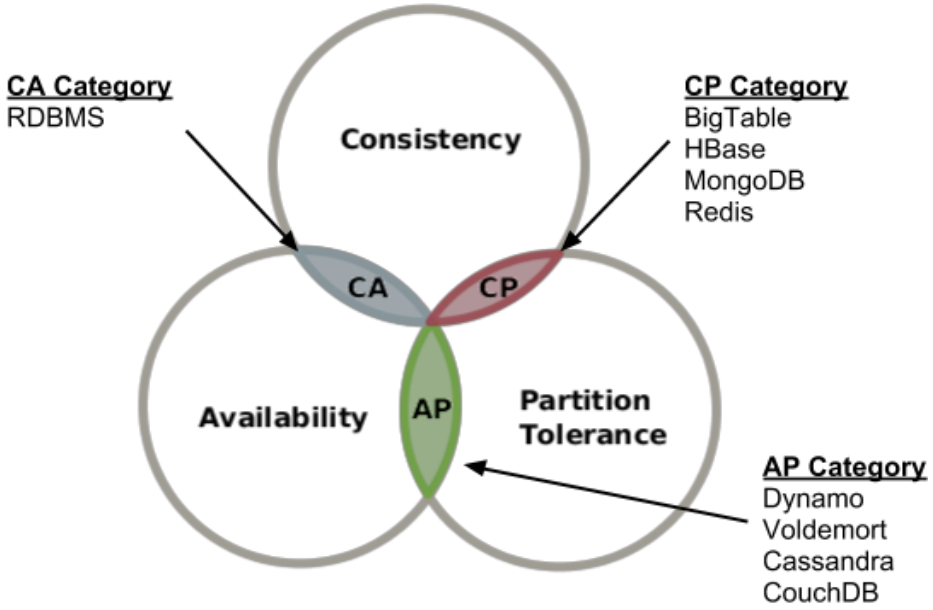
Assegurar o **ISOLAMENTO** de uma transação é **responsabilidade do Controle de Concorrência.**

# Propriedade das transações - ACID


- **Durabilidade**
  - Garante que quando uma transação tiver sido **confirmada** (***commit***), as atualizações dessa transação **não são perdidas**, mesmo que haja uma falha no sistema

# Transações: Teorema CAP

- Teorema criado para descrever o **comportamento de um sistema distribuído**
  - Propriedades
    - Consistência (***Consistency***)
      - Todas as réplicas contém a mesma versão do dado
    - Disponibilidade (***Availability***)
      - O SGBD deve continuar operacional após a ocorrência de falha em nós problemáticos
    - Tolerância à partição (***Partition Tolerance***)
      - O cluster pode suportar **falhas na comunicação** que o dividam em múltiplas partições incapazes de se comunicar



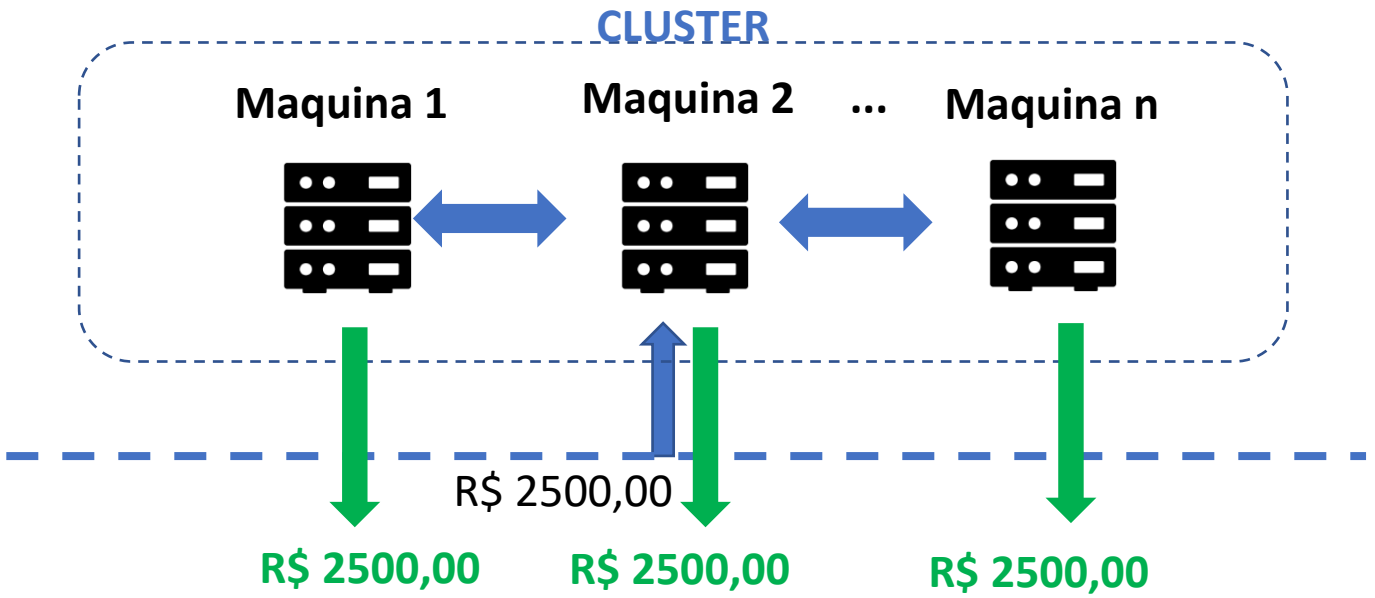

- Teorema
  - **é impossível garantir essas três propriedades ao mesmo tempo**

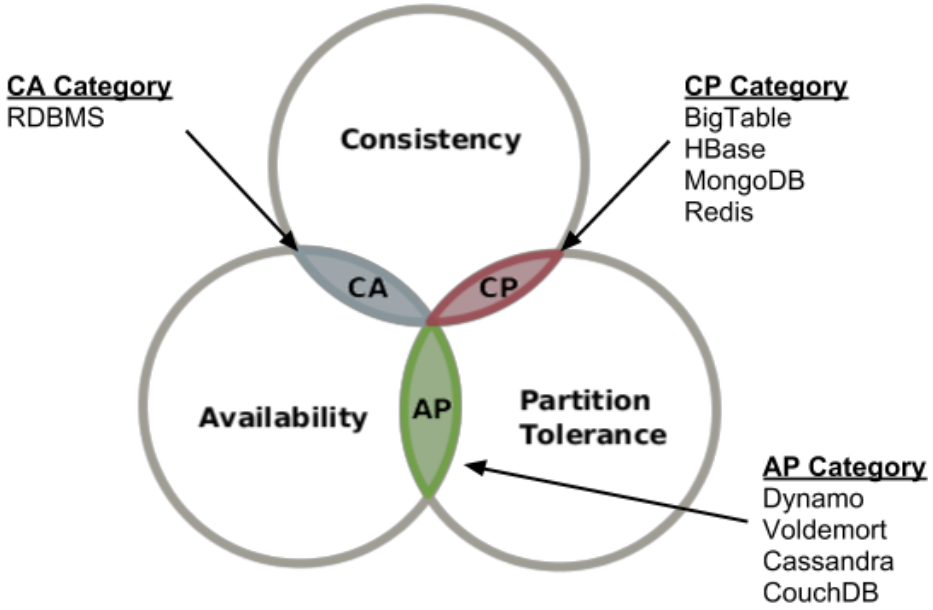
É possível garantir quaisquer duas dessas propriedades ao mesmo tempo

Gilbert, S.; Lynch, N. A. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51–59, 2002.

# Transações: Teorema CAP
# Consistência

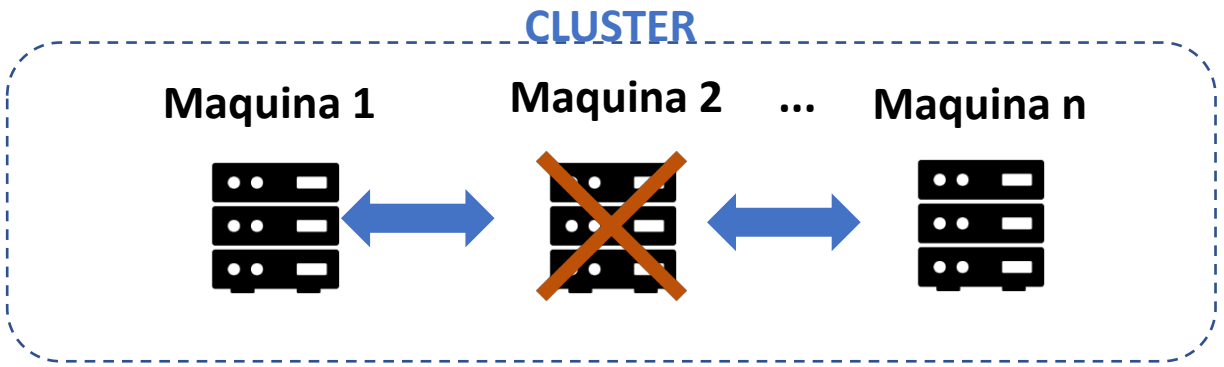
- Propriedades
  - Consistência (***Consistency***)
    - Todas as réplicas contém a mesma versão do dado

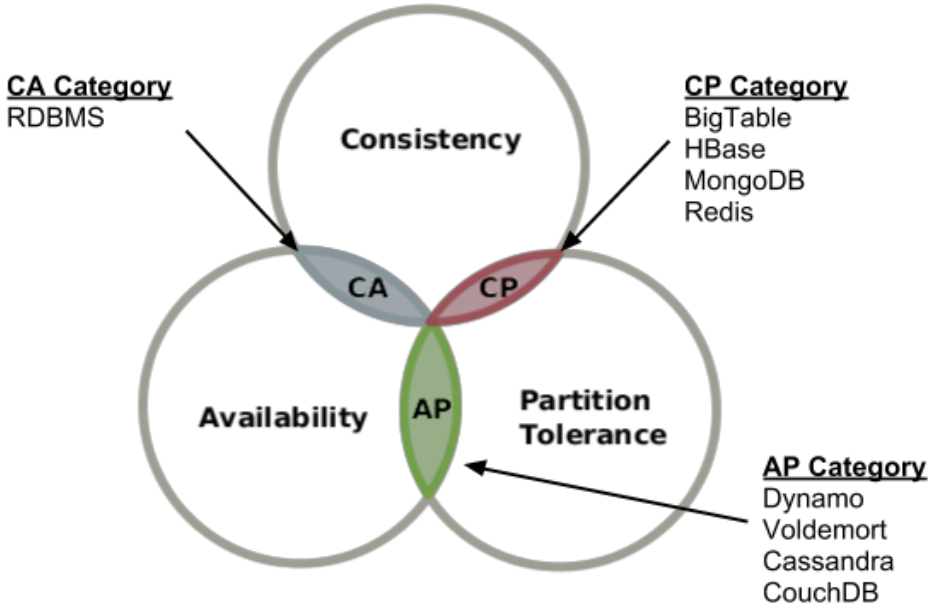


# Transações: Teorema CAP
# Disponibilidade

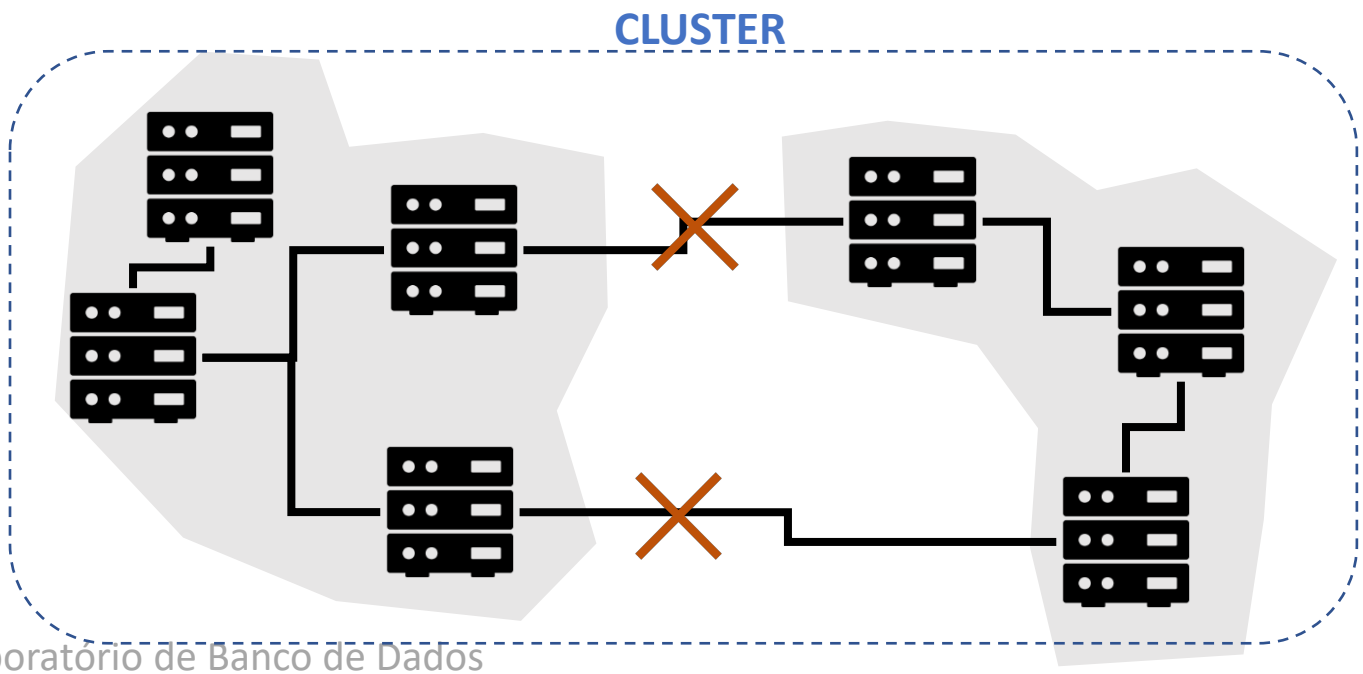
- Propriedades
  - Disponibilidade (***Availability***)
    - O SGBD deve continuar operacional após a ocorrência de falha em nós problemáticos

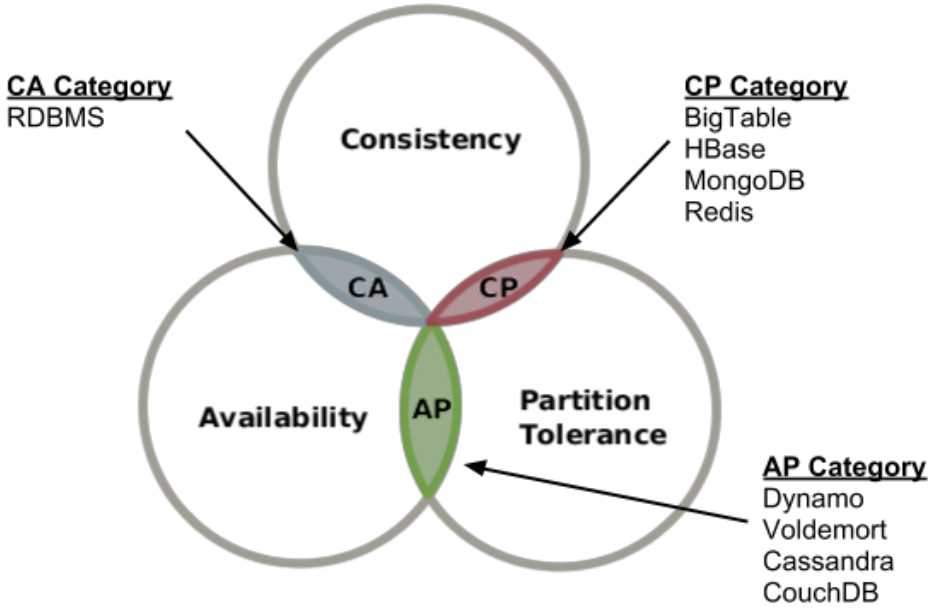


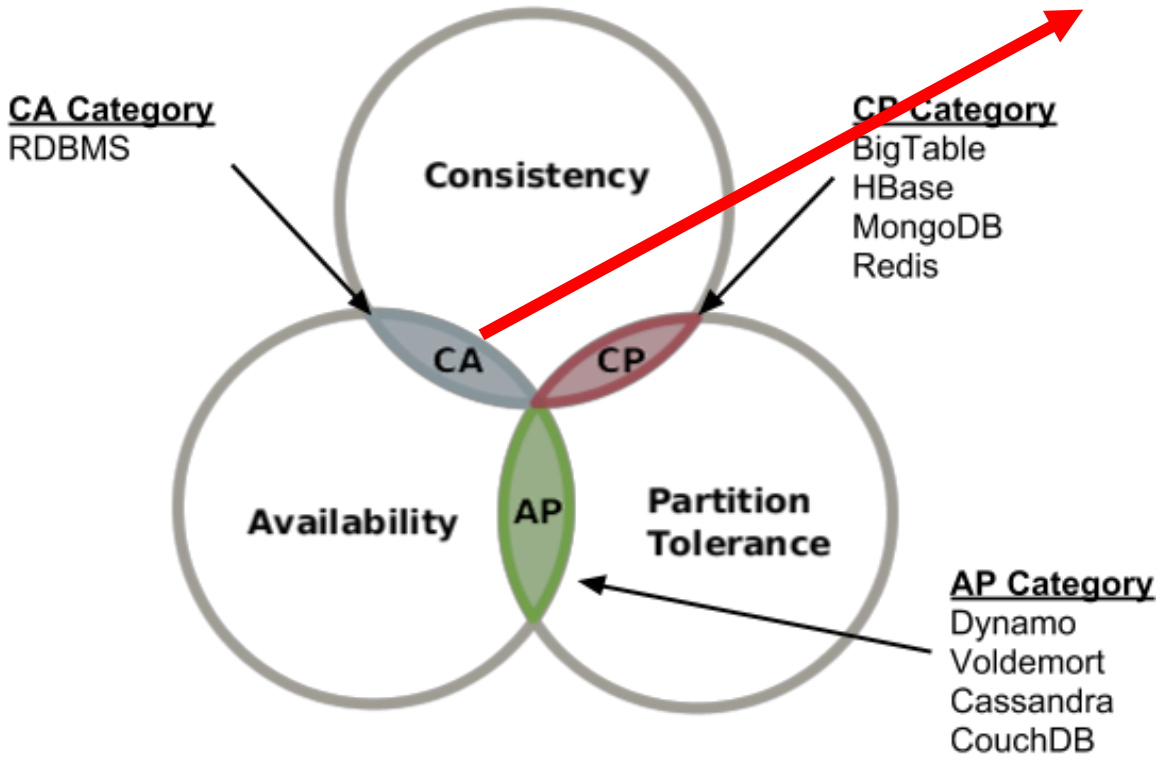
# Transações: Teorema CAP
# Tolerância a partição

- Propriedades
  - Tolerância à partição (***Partition Tolerance***)
    - O cluster pode suportar **falhas na comunicação** que o dividam em múltiplas partições incapazes de se comunicar
      - Múltiplos pontos de entrada
      - SGBD continua operacional após uma falha que ocasiona o particionamento do sistema



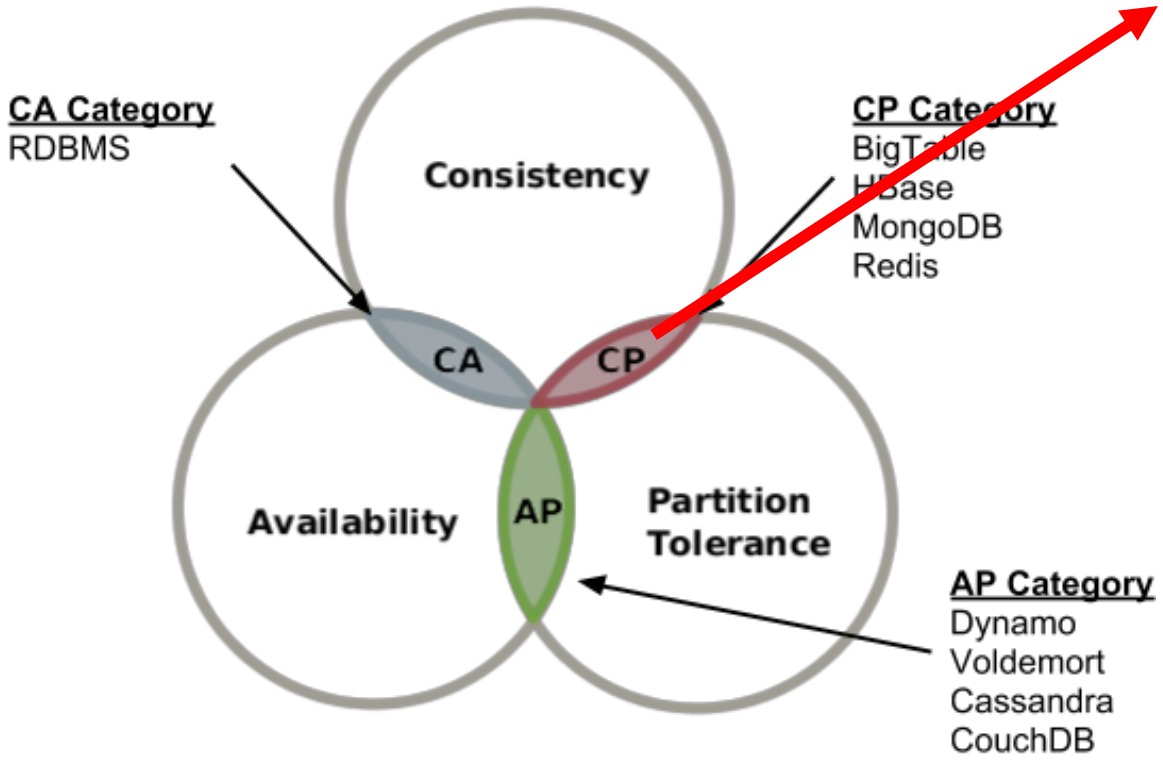



# Transações: Teorema CAP



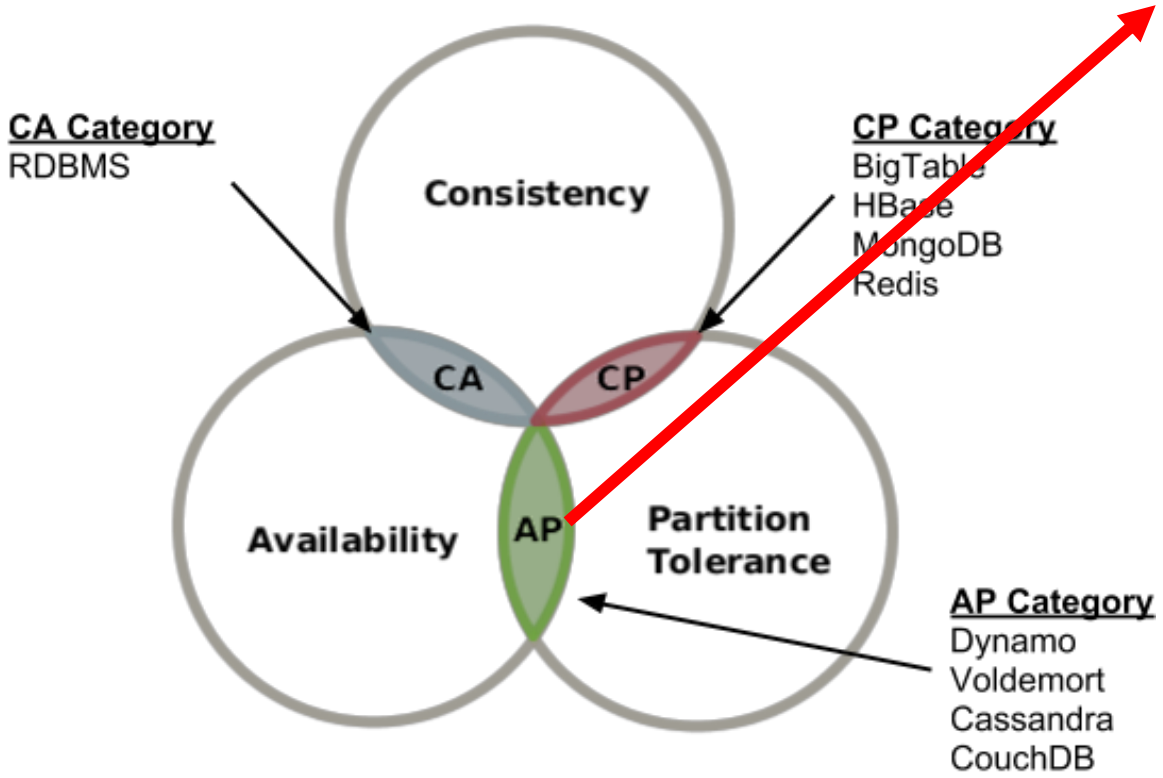

- **CA - Consistência e Disponibilidade**
  - Os sistemas com **consistência** forte e alta **disponibilidade (CA)** (alta disponibilidade de um nó apenas) **não sabem lidar com a possível falha de uma partição**.
  - **Problema: caso ocorra, sistema inteiro pode ficar indisponível até o membro do cluster voltar.**
  - **Exemplos disso são:**
    - **Uma única máquina não pode ser particionada, de modo que não é necessário preocupar-se com a partição**
      - **Maioria dos sistemas de banco de dados relacional**

# Transações: Teorema CAP




- **CP - Consistência e Tolerância**
  - Para sistemas que precisam da **consistência** forte e **tolerância a particionamento (CP)** é necessário abrir a mão da disponibilidade (*um pouco*).
  - **Problema: o sistema torna-se completamente inacessível se qualquer um dos nós estiver fora**

# Transações: Teorema CAP



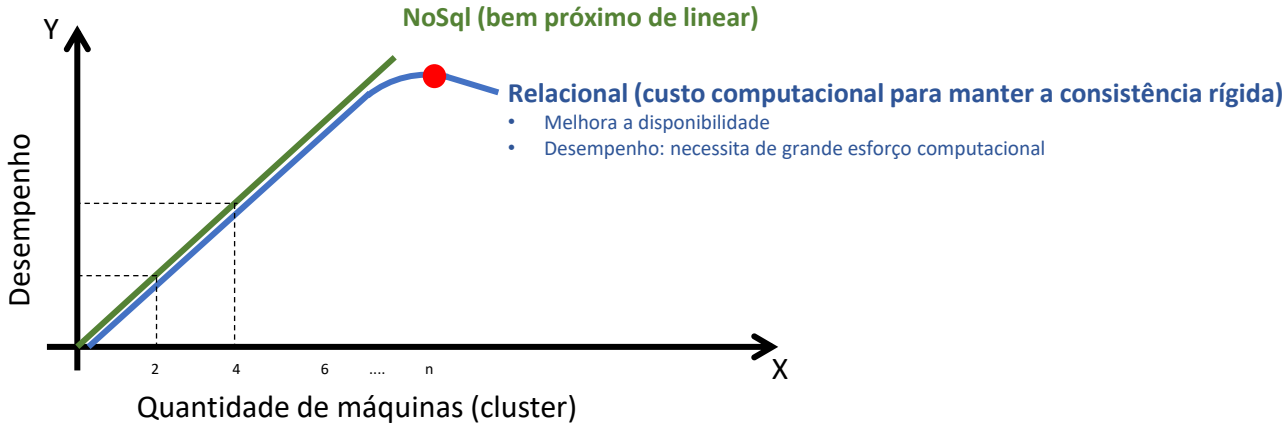

- **AP - Disponibilidade e tolerância**
  - Existem sistemas que jamais podem ficar *offline* (24/7), portanto não desejam sacrificar a disponibilidade.
  - Para ter alta **disponibilidade** mesmo com um **tolerância a particionamento (PA)** é preciso prejudicar a consistência (*eventual-consistency*). A ideia aqui é que os sistemas aceitam escritas sempre e tentam sincronizar os dados em algum momento depois (*read-consistency*).
  - **Problema: pode ter uma janela de inconsistência.**

# Transações

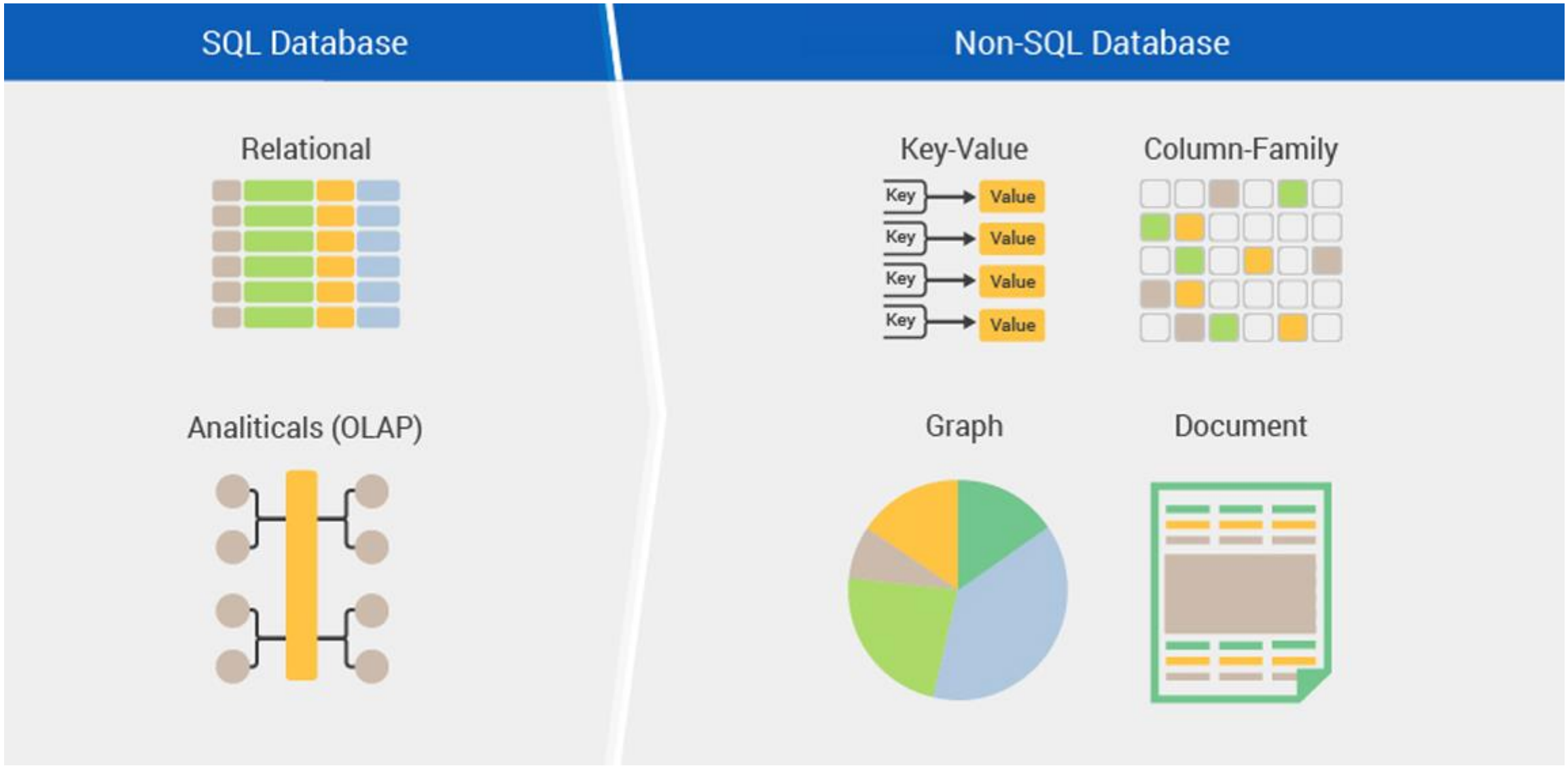
- Como consequência do teorema CAP as propriedades ACID não podem ser obedecidas simultaneamente para sistemas distribuídos
- Ao invés disso, para sistemas NoSQL foram descritas as propriedades **BASE**
  - ***Basically Available***
    - Algumas parte dos do sistema **continuam disponíveis após uma falha**
  - ***Soft-state***
    - A informação vai expirar a menos que a mesma seja atualizada. O sistema vai mudar o estado sem a intervenção do usuário devido a **consistência eventual**
  - ***Eventually concistent***
    - Propagação **assíncrona** dentro de uma janela de consistência (*consistancy window*)
    - Consequência: uma requisição de leitura logo após uma requisição de escrita pode ter como retorno o valor antigo

# Escolha do modelo relacional versus nosql

- Quando utilizar NoSQL?
  - Indicado para aplicações que:
    - Irão trabalhar com grande volume de dados
    - Necessitam de velocidade de resposta (leitura e escrita) – escala horizontal
    - Não necessita de consistência rígida
  - Ponto de inflexão



# Banco de dados NoSQL



# Banco de dados NoSQL

- Existem quatro categorias de modelos NoSQL, são eles:
  1. **Chave-valor**
  2. **DBMS orientado a grafos**
  3. **Armazenamento tabular**
  4. **DBMS orientado a documentos**


http://nosql-database.org/

# Banco de dados NoSQL
# Chave e valor


- **Chave-valor:** modelo simples onde cada registro tem uma chave única e um valor associado a essa chave.
  - Características
    - Representa dados por conjuntos de chaves e valores associados
      - Fácil implementação
      - Duas operações para manipulação: **PUT(chave,valor)** e **GET(chave)**
    - Proporciona bom desempenho para aplicações na nuvem
    - Oferece menor capacidade de busca, permitindo apenas busca por chave

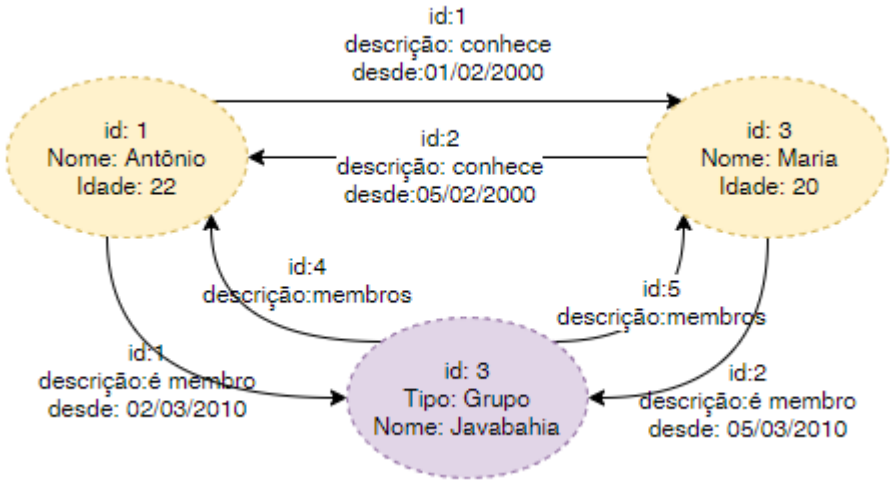

  - Aplicações típicas
    - Gerenciamento de sessões
    - Perfis e preferências dos usuários
  - Ruim
    - Relacionamento entre dados
    - Operações a partir de conjuntos

# Banco de dados NoSQL

Grafo

Neo4j the graph database | InfiniteGraph | sones

- **Grafos:** os dados são armazenados em nós de um grafo, esses bancos são úteis para modelar e acessar dados com alto grau de complexidade
- Características
  - Baseados na teoria dos grafos
  - Dados são modelados por nós e arestas, os quais podem possuir propriedades
- Aplicações típicas
  - Redes sociais
  - Recomendações

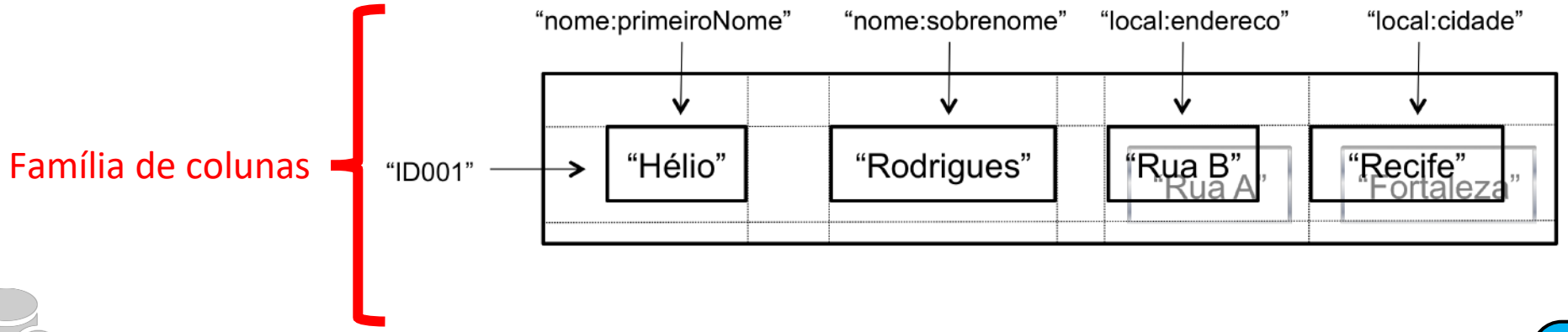


# Banco de dados NoSQL

Tabular

HBASE riak Cassandra

- **Tabular ou Família de colunas:**
  - Utiliza tabelas gigantes onde as chaves apontam para várias colunas distintas
  - Serve para armazenar grandes quantias de dados em máquinas diferentes
- Características
  - Cada coluna é definida por uma tripla
    - **chave**, **valor** e ***timestamp***
      - Onde linhas e colunas são identificadas por chaves e o *timestamp* permite diferenciar múltiplas versões de um mesmo dado



# Banco de dados NoSQL


- **Documentos:** cada registro é representado por um documento composto por elementos e sub-elementos, o documento é uma estrutura maleável e dinâmica, pode ser alterado a qualquer momento
- Características
  - Cada documento é uma coleção de pares chave-valor
  - Usualmente baseado nos formatos XML ou JSON
  - Permite a realização de consultas mais elaboradas, envolvendo filtros por atributos e a possibilidade de uso de índices
- Aplicações típicas
  - Aplicações Web
  - Registro de log
- Ruim
  - Consultas agregadas

# JSON – JavaScript Object Notation

- Formato leve para troca de informações
- Para seres-humanos
  - Fácil leitura e compreensão
- Para máquinas
  - Fácil de construir e interpretar
- Formato independente de linguagem
- Comparado a XML
  - Mais fácil e rápido de construir

```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ]
```

# JSON – JavaScript Object Notation

- Duas estruturas básicas:
  - Uma coleção de pares nome/valor
    - Em outras linguagens: struct, record, objeto, tabela hash, etc.
      - **Ex.: { "pessoa" : “Ana” }**
  - Uma lista ordenada de valores
    - Em outras linguagens: arrays, listas, sequência, vector, etc.
      - **Ex.: [ "Futebol", "Basquete", "Volei", "Natação" ]**

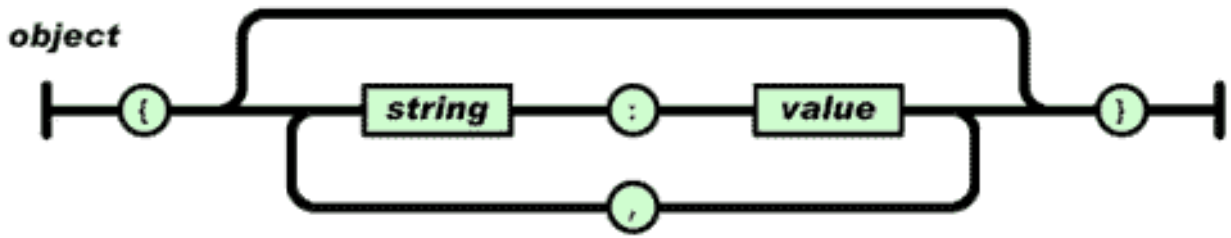
```
{ "Books":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course' - a great deal!",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  -
```

# JSON – JavaScript Object Notation

- Estruturas JSON são universais
  - Podem ser encontradas em praticamente todas as linguagens
  - É o que permite o intercâmbio de informações
- O formato JSON é idêntico ao formato de objetos de Javascript
  - Pode ser facilmente criado e manipulado em Javascript
  - Não há necessidade de software extra para trabalhar com JSON em Javascript
- Estruturas de dados JSON
  - Objeto
  - Array
  - Valor
  - Número
  - String

# JSON – Objeto

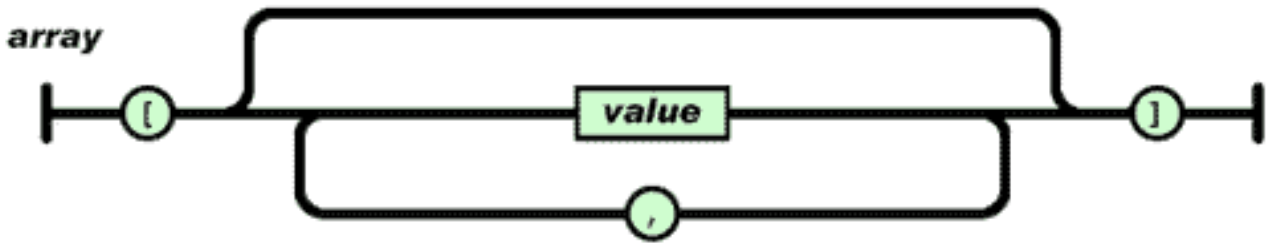
- OBJETO
  - Conjunto de pares **nome**/**valor**
  - Começa com **{** e termina com **}**
  - Cada nome é seguido por “**:**” (dois pontos)
  - Pares são separados por “**,**” (vírgula)




```
{"Nome": "Ana",
    "Sobrenome": "Jones",
    "Gênero": "F",
    "Idade": 40 }
```

# JSON – Array

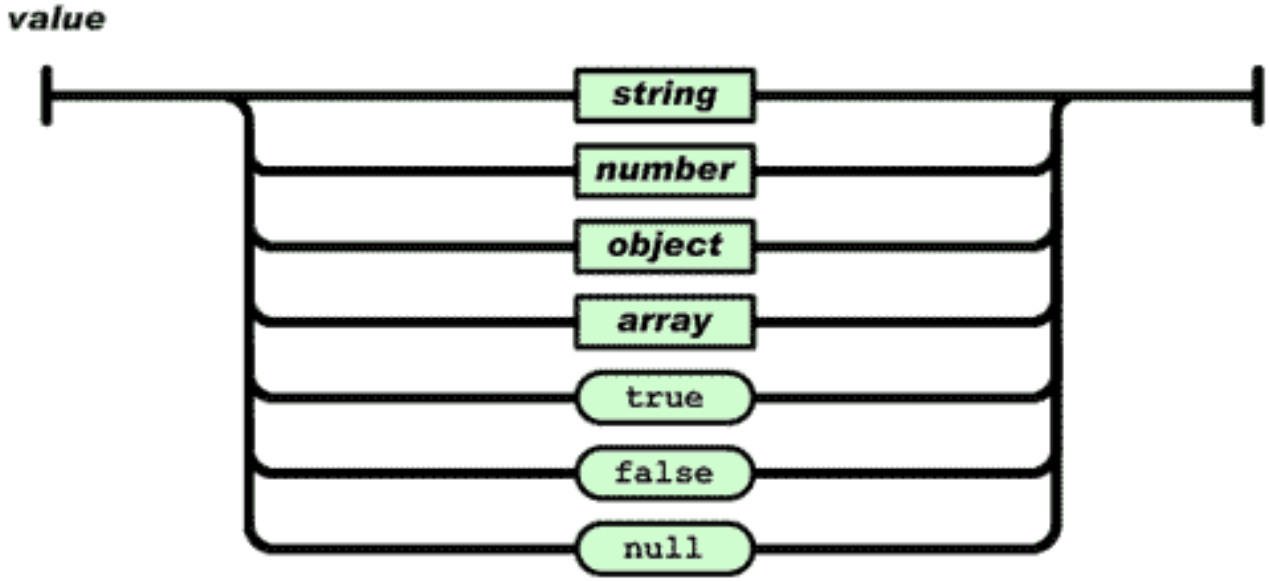
- ARRAY
  - Coleção ordenada de valores
    - Começa com [ e termina com ]
    - Valores são separados por “,” vírgula




```
{"Funcionário":
    [
        {"nome": "John", "sobrenome": "Doe"}
        {"nome": "Ana", "sobrenome": "Smith"}
        {"nome": "Piter", "sobrenome": "Jones"}
    ]
}
```

# JSON – Valor

- VALOR
  - Pode ser uma string, número, objeto, array ou as palavras NULL, TRUE e FALSE




Exemplo de string

Exemplo de número

Exemplo de booleano

```
{"nome": "Ana"}
{"altura": 1.72}
{"temperatura": -2}
{"casado": true}
```

# Banco de dados orientado a documentos

# conceitos básicos

# Principais diferenças entre relacional e orientado a documentos

| Banco de dados Relacional | | MongoDB |
|---|---|---|
| Armazenam dados em tabelas com linhas divididas em colunas | → | Armazenam os dados em coleções de documentos com campos e subdocumento |
| Curva de aprendizado maior | → | Curva de aprendizado menor |
| Possuem relacionamento entre diferentes tabelas e registros | → | Não possuem relacionamentos entre coleções e/ou documentos |
| Foco em garantir o ACID | → | Foco em garantir escalabilidade, alta-disponibilidade e performance |
| Uso de transações para garantir commits e rollbacks | → | Não há uso de transações e o ACID é garantido apenas a nível de documento. Transações devem ser garantidas a nível de aplicação |
| Uso da linguagem de consulta SQL para manipular o banco | → | Uso de comandos JavaScript para manipular o banco |

# Principais diferenças entre relacional e orientado a documentos

| Banco de dados Relacional | | MongoDB |
|---|---|---|
| Cada coluna de uma linha pode armazenar apenas um dado (atômico) | → | Cada campo de um documento pode armazenar múltiplos valores ou até mesmo outros documentos |
| Possuem chave-estrangeira e JOINs | → | Não possuem chave-estrangeira nem JOINs |
| Schema pré-definido e rígido | → | Schema variável conforme o uso da aplicação |
| Foco em não repetição de dados | → | Foco em acesso rápido de dados |
| Menor consumo de disco | → | Maior consumo de disco (geralmente) |
| Menor consumo de memória | → | Maior consumo de memória (geralmente) |

# O que é MongoDB?

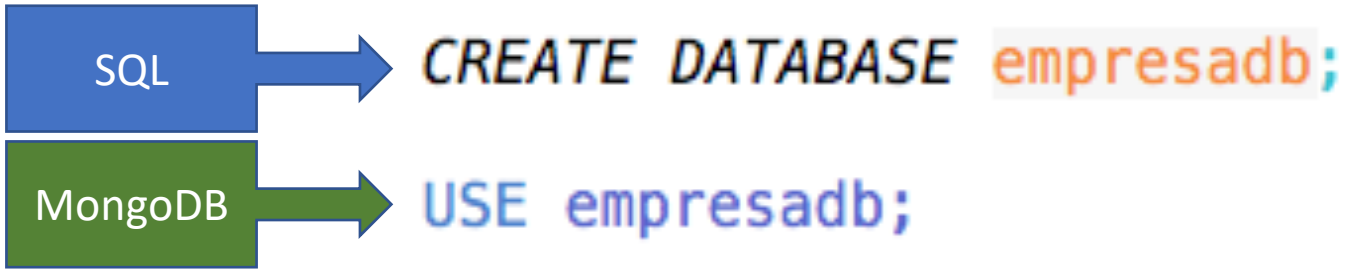
- Alguns de seus diferenciais são:
  - Alto desempenho
    - documentos **embutidos** e **índices** atuando sobre eles
  - Rica linguagem de consulta
    - permite operações **CRUD**, **agregações** de dados, busca por **texto** e consultas
  - Alta disponibilidade: ***replica set***
    - Replicar dados entre instâncias permitindo maior tolerância a falhas
  - Escalabilidade horizontal: ***sharding***
    - Dividir os dados (e a carga de I/O) entre instâncias (ou entre replica set) permitindo maior performance em leitura e escrita

# O que é MongoDB?

- O MongoDB é um **SGBD NOSQL *open source*** e **orientado a documentos**
  - Um registro no MongoDB é um documento, que é uma estrutura de dados composta de pares de chave e valores.
  - Os documentos do MongoDB são semelhantes aos objetos JSON.
    - Os valores dos campos podem incluir outros documentos, matrizes e matrizes de documentos

# Criando um Banco de Dados

- O MongoDB **abstrai** diversos comandos **DDL**.
  - Estruturas são criadas conforme estas se tornam necessárias.
- Para criar um banco de dados, basta você usar o comando para **acessar um banco que ainda não existe**.
  - Assim que um registro for inserido neste banco, ele será **criado e persistido automaticamente**.



# Coleções e documentos

- Como em outros modelos orientado a documentos, o MongoDB organiza os dados em **coleções** de **documentos**
  - Cada documento possui um atributo identificador (_id) e uma **quantidade qualquer** de outros atributos
  - **Não é necessário (mas é possível) especificar o ID dos documentos**
  - **Não é necessário (mas é possível) especificar o tipo dos atributos**
  - **Documentos diferentes que fazem parte de uma mesma coleção podem ter atributos diferentes**

# Operações no MongoDB - CRUD

**INSERÇÃO | LEITURA | ALTERAÇÃO | REMOÇÃO**

# Operações no MongoDB - Inserção

- Inserção
  - Para criar uma coleção, basta **inserir um documento nela**

db.COLLECTION_NAME.insert (<documento | array de documentos>)

```
db.cliente.insert(
    {nome:"Ana",
    endereco:"Rua Arruda Alvin"});

db.cliente.insert([
    {nome:"João", endereco:"Rua Sabiá"},
    {nome:"Renato", endereco:"Rua Jacutinga"}]
);
```

# Operações no MongoDB - Inserção

SQL

```sql
CREATE TABLE cliente (
id integer NOT NULL,
nome varchar(255) NOT NULL,
endereco varchar (255)
);

INSERT INTO cliente (id, nome, endereco)
VALUES (1, 'Margarida', 'Rua Oscar Freire');
```

MongoDB

```
db.cliente.insert(
    {nome:"Margarida",
    endereco:"Rua Oscar Freire"});
```

No mongoDB você não precisa criar a coleção, pois o MongoDB cria a coleção automaticamente quando você insere algum documento.

# Operações no MongoDB - Inserção

```
db.estoque.insert({
    item:"chocolate",
    detalhes: {
        fabricante:"nestle",
        modelo:1234},
    itensestoque:[
        {qtd:50, tamanho:"P"},
        {qtd:20, tamanho:"M"}],
    categoria:"alimento"
});
```

```
db.estoque.find();
```

estoque 0.001 sec.

```
/* 1 */
{
    "_id" : ObjectId("5bd4fc2813e3745a2fef5049"),
    "item" : "chocolate",
    "detalhes" : {
        "fabricante" : "nestle",
        "modelo" : 1234.0
    },
    "itensestoque" : [
        {
            "qtd" : 50.0,
            "tamanho" : "P"
        },
        {
            "qtd" : 20.0,
            "tamanho" : "M"
        }
    ],
    "categoria" : "alimento"
}
```

Documento

Documento

Array de documentos

# Operações no MongoDB - Atualização

- O comando update() atualiza os valores

db.COLLECTION_NAME.update(<query>, <update>, <options>)

SQL

```
UPDATE cliente SET nome'Meg' WHERE NOME='Margarida';
```

MongoDB

```
db.cliente.update({"nome":"Margarida"},{$set:{"nome":"Meg"}},{upsert:true})
```

Se a condição de seleção da query (filtro) não for encontrada, o documento será inserido na coleção.

# Operações no MongoDB - Atualização

- Adicionando um atributo multivalorado (array) a um documento da coleção

```
db.cliente.update(
    {_id : ObjectId("5bd4ede513e3745a2fef5044")},
    {$set:{telefones:[123456, 765432]}})
```

```
{
    "_id" : ObjectId("5bd4ede513e3745a2fef5044"),
    "nome" : "Margarida",
    "endereco" : "Rua Oscar Freire",
    "telefones" : [
        123456.0,
        765432.0
    ]
}

/* 2 */
{
    "_id" : ObjectId("5bd4ede513e3745a2fef5045"),
    "nome" : "Ana",
    "endereco" : "Rua Arruda Alvin"
}
```

# Operações no MongoDB - Remoção

- Remove um documento de uma coleção

db.COLLECTION_NAME.remove(<query>, <qtd documentos>)

SQL

```
DELETE FROM funcionario WHERE nome = 'Carlos';
```

MongoDB

```
db.funcionario.remove({nome:"Carlos"})
db.funcionario.remove({nome:"Carlos"},1)
```

# Operações no MongoDB - Seleção

- Seleção
  - O MongoDB possui dois métodos principais para retornar informações de documentos

```
db.COLLECTION_NAME.find(<query>, <projection>)
db.COLLECTION_NAME.findOne(<query>, <projection>)
```

    - O método **find()** retorna **um ponteiro para todos** os documentos que atendem aos critérios especificados
    - O método **findOne()** retorna **um único documento** que atende aos critérios especificados
      - Caso exista mais de um documento atendendo aos critérios, o método **findOne()** retorna **apenas o primeiro**

# Operações no MongoDB - Seleção

- Seleção
  - O método **find()** permitem especificar, **da mesma forma**, critérios de **seleção** e **projeção** para o resultado.

SQL

```
SELECT nome FROM cliente WHERE nome='Margarida';
```

Projeção — Seleção

MongoDB

```
db.cliente.find({nome:"Margarida"}, {nome:1})

db.cliente.find({nome:"Margarida"}, {_id:0, endereco:0})
```

Seleção — Projeção

O método find() mostra todos os campos.
Para ocultar campos ou mostrar campos pode-se utilizar como parâmetro 0|false (ocultar) e 1|true (mostrar).

# Operações no MongoDB – Seleção - IN

SQL

```
SELECT * from cliente
WHERE nome IN ('Margarida', 'Ana');
```

MongoDB

```
db.cliente.find({nome:
    {$in:
    ["Margarida", "Ana"]}
})
```

```
db.cliente.find({nome:{$in: ["Margarida", "Ana"]}})
```

cliente 0.001 sec.

```
/* 1 */
{
    "_id" : ObjectId("5bd4ede513e3745a2fef5044"),
    "nome" : "Margarida",
    "endereco" : "Rua Oscar Freire",
    "telefones" : [
        123456.0,
        765432.0
    ]
}

/* 2 */
{
    "_id" : ObjectId("5bd4ede513e3745a2fef5045"),
    "nome" : "Ana",
    "endereco" : "Rua Arruda Alvin"
}
```

# Operações no MongoDB – Seleção - Operadores

SQL

```
SELECT * FROM funcionario WHERE
    salario >= 1000 AND salario <= 5000
    AND sexo='M';
```

MongoDB

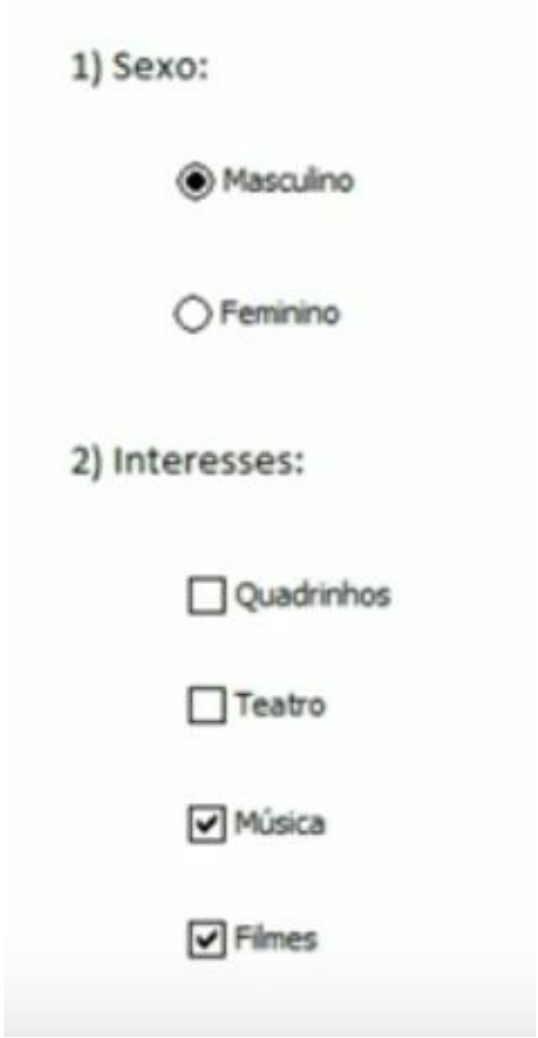
```
db.funcionario.find({
    salario:{$gte:1000, $lte:20000}, $and:[{sexo:"M"}]
})
```

```
db.funcionario.find({
        salario:{$gte:1000, $lte:20000}, $and:[{sexo:"M"}]
})
```

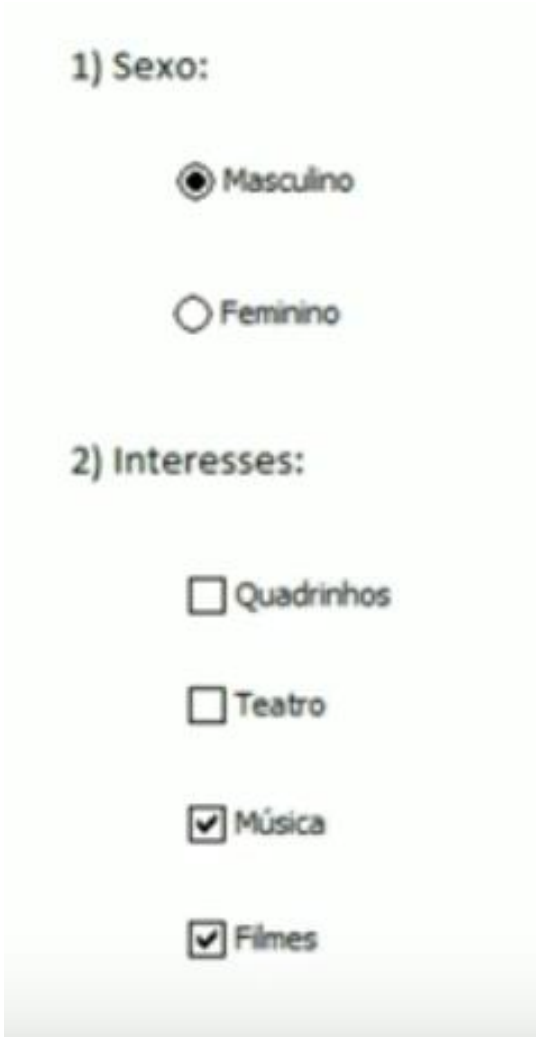
funcionario 0.001 sec.

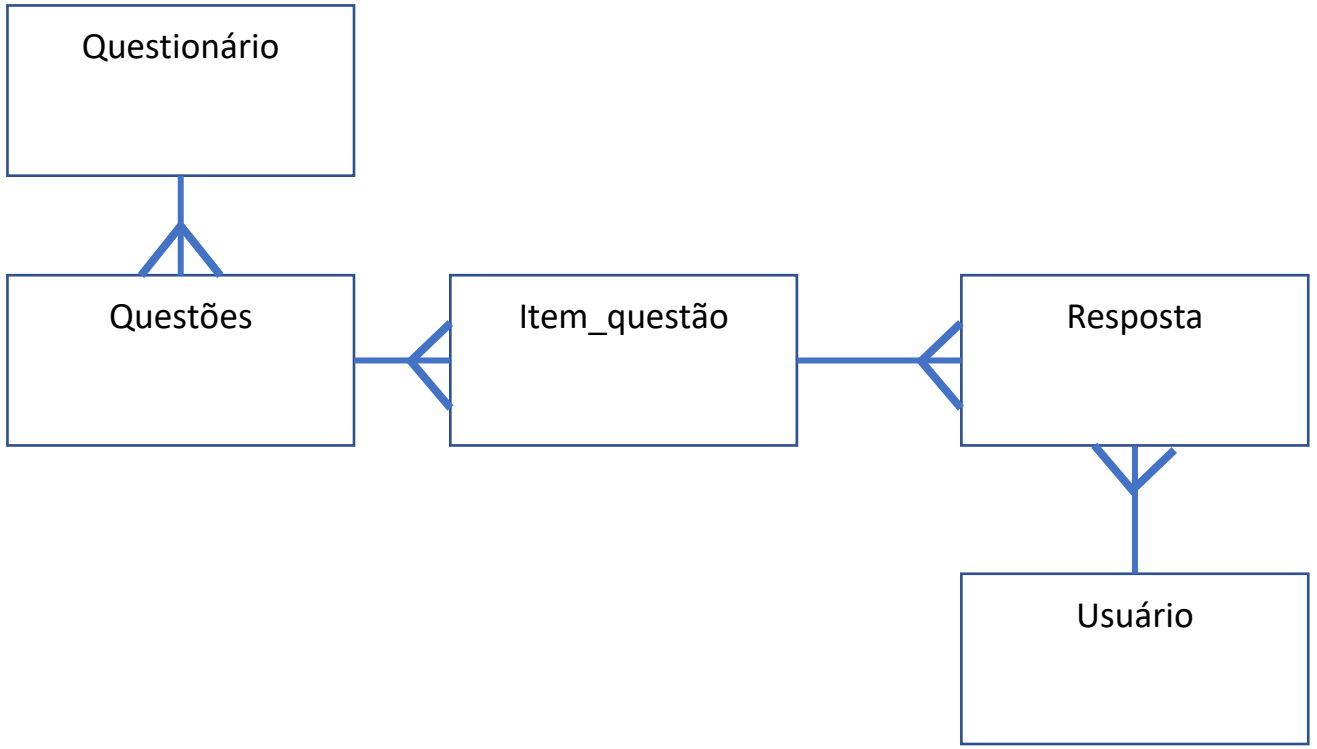
```
/* 1 */
{
    "_id" : ObjectId("5bd50d7413e3745a2fef504c"),
    "nome" : "Francisco",
    "salario" : 12000.0,
    "sexo" : "M"
}
```

# Como seria essa modelagem?

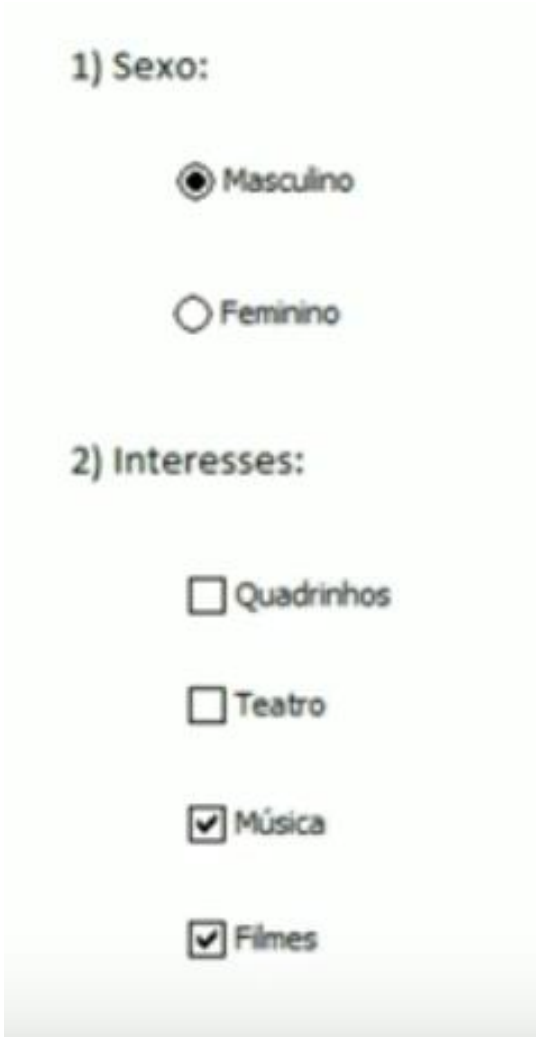


# Como seria essa modelagem?



# Como seria essa modelagem?




```
use dbquestionario

db.questionario.insert(
    codusuario:'USR001',
    nome: 'José da Silva',
    questões: {q01:'Masculino', q02: 'Musica'})
```

# Relacionamento no MongoDB

- O MongoDB **não implementa integridade referencial e nem operações de junção**
  - Logo, **não existe** o conceito de **chave estrangeira** para documentos
- Existem duas maneiras de se expressar relacionamentos entre documentos no MongoDB
  - **Referencias entre documentos:** é possível guardar o **_id** de um documento como um atributo em outro documento
    - **Não é o mesmo que guardar uma chave estrangeira**
  - **Documentos embutidos:** o MongoDB permite guardar um **documento inteiro** como um atributo em um documento (Subdocumentos)

# Relacionamento entre coleções

- Supondo de uma escola que possui uma coleção de alunos e uma coleção de livros na biblioteca.
  - Deseja-se controlar o empréstimo dos livros

```
db.aluno.insert([
    {_id:1, nome: "Carolina", sobrenome:"Alves"},
    {_id:2, nome: "Antonio", sobrenome:"Correia"},
    {_id:3, nome: "Cassio", sobrenome:"Moreira"}])

db.livro.insert([
    {_id:1, titulo: "Programação orientada a objetos", autor: "Barnes"},
    {_id:2, titulo: "Redes de computadores", autor:"Tanenbaum"},
    {_id:3, titulo: "Sistemas de banco de dados", autor:"Navathe"}])
```

# Relacionamento entre coleções

- Como controlar o empréstimo:

```
-- Controlando o empréstimo
db.livro.update({_id:1},
    {$set:
        {aluguel:{aluno_id:2,
            periodo:{
                inicio:"11/10/2018",
                fim:"15/10/2018"}}
}})

-- Livros emprestados
db.livro.find({aluguel:{$exists:true}})

-- Livros disponíveis
db.livro.find({aluguel:{$exists:false}})
```

```
db.livro.find()

livro  0.001 sec.

/* 1 */
{
    "_id" : 1.0,
    "titulo" : "Programação orientada a objetos",
    "autor" : "Barnes",
    "aluguel" : {
        "aluno_id" : 2.0,
        "periodo" : {
            "inicio" : "11/10/2018",
            "fim" : "15/10/2018"
        }
    }
}

/* 2 */
{
    "_id" : 2.0,
    "titulo" : "Redes de computadores",
    "autor" : "Tanenbaum"
}

/* 3 */
{
    "_id" : 3.0,
    "titulo" : "Sistemas de banco de dados",
    "autor" : "Navathe"
}
```

# Comandos úteis

- Lista todos os databases
  - Show dbs
- Conecta em um database
  - Use <nome do banco de dados>
- Lista todas as collections
  - Show collections